VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webové frameworky v jazyce Scala

Web frameworks in Scala

2013

Lukáš Zemek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Zemek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Webové frameworky v jazyce Scala
Web Frameworks in Scala

Zásady pro vypracování:

Cílem práce je vytvořit přehledovou studii webových aplikačních rámců v jazyce Scala.

Student se zaměří na popis frameworku Lift, a alespoň jednoho dalšího, provede srovnání vybraných frameworků a implementuje ukázkovou aplikaci, na které bude demonstrovat silné a slabé stránky vybraných frameworků.

Seznam doporučené odborné literatury:

The Definitive Guide to Lift: A Scala-based Web Framework (Expert's Voice in Open Source) Marius Danciu (Author), Tyler Weir (Author), Derek Chen-Becker (Author)

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Krömer, Ph.D.**

Datum zadání: 19.11.2010
Datum odevzdání: 06.05.2011

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. 4. 2013 .....

Rád bych na tomto místě poděkoval Ing. Pavlu Krömerovi, Ph.D. za vstřícný přístup a konzultace během vypracovávání této bakalářské práce a rovněž mým rodičům, kteří mě neustále podporovali a bez kterých by tato práce nevznikla. Práce samotná byla vysázena v systému LaTeX.

Abstrakt

V této práci vytvářím přehledovou studii webových frameworků v programovacím jazyce Scala. V první části práce popisuji jednotlivé vlastnosti jazyka Scala, v části druhé se zaměřuji na Lift a jeho popis. V další části uvádím dokumentaci k mnou navržené webové aplikaci ve frameworku Lift a zobrazuji diagram návrhu systému. V následující části popisuji další framework a tím je Play, u kterého také vytvářím ukázkovou aplikaci. V závěrečné části práce porovnávám výhody a nevýhody těchto dvou frameworků a zhodnocuji přínos celé práce.

Klíčová slova: Scala, Lift, Play, Maven

Abstract

In this work I create a study overview of web-application frameworks in the Scala language. In the first part I describe individual properties of Scala language, in the second part I focus on the Lift framework and its description. In the next part I mention the documentation to my designed web application in the Lift framework and show a chart of system design. In the next part I describe another framework Play, where I also create a web application. In the final section I compare advantages and disadvantages of these two frameworks and evaluate the benefits of the whole work.

Keywords: Scala, Lift, Play, Maven

Seznam použitých zkratek a symbolů

XML	–	Rozšiřitelný značkovací jazyk (Extensible Markup Language)
HTTP	–	Protokol pro výměnu hypertextových dokumentů (Hypertext Transfer Protocol)
JVM	–	Virtuální stroj Javy (Java Virtual Machine)
UML	–	Unifikovaný modelovací jazyk (Unified Modeling Language)
JDBC	–	Spojení k Java databázi - Java Database Connectivity
AJAX	–	Asynchronní JavaScript a Xml - Asynchronous JavaScript and Xml

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Když je řeč o webových frameworcích, musíme definovat, co takový *webový framework* [1] (nebo také *aplikační rámec*) vlastně je. Jedná se o softwarový rámec, který je navržen pro podpoře vývoje dynamických webových stránek, webových aplikací a také webových služeb. Webový framework se zaměřuje na zmírnění režie spojenou s běžnými aktivitami spojenými s vývojem webových aplikací. Například mnoho frameworků nabízí knihovny pro přístup k databázi, vytváření šablon aplikací a také správu sezení (z angl. *sessions*) a většinou se snaží o možnost znovupoužití daného kódu, který byl již někdy napsán.

V následujících kapitolách přiblížím vlastnosti jednotlivých frameworků jazyka Scala a to zejména frameworku Lift, který má v současné době největší podporu mezi programátorskou komunitou.

2 Scala

Název Scala [2, 4] značí „škálovatelný jazyk“ (v originále *scalable language*). Jazyk byl takto pojmenován, protože byl zkonstruován k tomu, aby rostl, rozšiřoval a vyvíjel se současně s požadavky uživatele. Technicky vzato se ve Scale prolínají principy objektového a funkcionálního programování, avšak ve staticky typovaném jazyce. Druhý význam můžeme hledat v italském překladu slova „scala“, který znamená schody.

2.1 Historie a vývoj

Autorem Scaly je Martin Odersky z *École Polytechnique Fédérale de Lausanne* (EPFL) ve Švýcarsku, který již před tím pracoval na Javě: byl spoluautorem překladače javac a rozšíření (superset) Generic Java. Vývoj Scaly začal v roce 2001. První verze pro platformu Java vyšla na konci 2003/začátku 2004, pak za půl roku v červnu 2004 uviděla svět první verze pro platformu .NET. Poslední verze v době psaní této práce je 2. 8. 0.

2.2 Vlastnosti jayzka Scala

Jak jsem již zmínil výše, Scala je objektově orientovaný programovací jazyk pro JVM a také jazyk funkcionální. Tato kombinace přístupů nám dovoluje využít moderní přístupy k programování a rovněž nám ponechává možnost použít všechen již existující kód Javy.

Scala usiluje o co nejkratší kód. V porovnání s Javou by při správném použití konstrukcí jazyka mohl být přibližně o polovinu kratší. Méně řádků kódu neznamená jen méně psaní, ale i lepší přehlednost a srozumitelnost programů. Také méně možností, kde by mohla nastat chyba.

Ve Scale můžeme bez problémů používat všechny třídy z klasického Java API. To nám ukazuje vysokou interoperabilitu mezi jazykem Java a Scalou.

2.3 Proměnné

Ve Scale rozlišujeme proměnné dvou typů:

- `val` – odpovídá final proměnným v Javě, tedy konstanta (po inicializaci nelze změnit hodnotu proměnné)
- `var` – odpovídá non- final proměnným v Javě, tedy hodnota proměnné může být později změněna

2.4 Funkce

Definice funkcí začínají klíčovým slovem `def`. Následuje jméno funkce a poté v závorkách uvedené jednotlivé parametry. Každý parametr funkce musí následovat typová anotace, neboť kompilátor a interpretr Scaly si sám neodvodí typy parametrů funkcí. Za ukončenou závorkou a dvojtečkou je uveden *výsledný typ* funkce samotné [1]. Poté následuje tělo funkce. Ve výpise 1 je uveden příklad funkce.

Pokud je funkce rekurzivní [2], tak musíme explicitně specifikovat výsledný typ. Jakmile máme funkci vytvořenu, můžeme ji volat podle jména.

```
scala> def max(x: Int, y: Int): Int = {
         if (x > y) x
         else y
       }

max: (x: Int, y: Int)Int
```

Výpis 1: Ukázka jednoduché funkce

```
scala> def greet() = println("Hello, world!")
greet: ()Unit
```

Výpis 2: Definování funkce

greet- název funkce *()*- značí, že funkce nepřebírá žádné parametry. *Unit*- značí, že funkce nevrací žádnou důležitou hodnotu. Návratová hodnota Unit v jazyku Scala je podobná návratové hodnotě *void* v jazyce Java.

2.5 Rysy

Rysy (z angl. *traits*) jsou stejné jako rozhranní v Javě, ale mohou obsahovat implementaci metod. Nemohou přebírat parametry konstruktoru, ale jinak se chovají jako třídy. To nám dává možnost mít k dispozici něco na způsob vícenásobné dědičnosti, aniž bychom museli řešit vznik Diamantového problému (z angl. *Diamond problem*) [15].

V definici rysu používáme klíčové slovo `trait`. Jakmile je rys definován, může být „zakomponován“ do třídy tak, že použije klíčové slovo `extends` nebo `with`.

2.6 Pole a seznamy

Jednou z hlavních myšlenek funkcionálního programování je to, že by metody neměli mít vedlejší účinky. Tedy že by měly zpracovat výpočet a vrátit hodnotu. Pokud budeme k metodám přistupovat tímto stylem, budou méně spletené a tedy i více spolehlivé a znovupoužitelné.

[1] V Javě se typ hodnoty vrácený metodou nazývá návratový typ, ve Scale je stejnému konceptu říká výsledný typ (z angl. *result type*).

[2] Tedy funkce volá sama sebe.

Pro neměnnou sekvenci objektů stejného typu můžeme ve Scale použít třídu *List*. Stejně jako u polí, seznam *List[String]* bude obsahovat pouze a jen řetězce. Třída *scala.List* se liší od *java.util.List* použité v Javě tím, že seznamy ve Scale jsou vždy neměnné (naproti tomu seznamy v Javě mohou být proměnlivé). Seznamy ve Scale jsou tedy navrženy k tomu, aby umožnily funkcionální styl programování.

2.7 N-tice

K dalším užitečným kontejnerem objektů patří n-tice (z angl. *tuples*). Stejně jako seznamy, i n- tice jsou neměnné. Ale narozdíl od seznamů, n- tice mohou obsahovat elementy různých typů.

```
val pair = (99, "Luftballons")
println(pair._1)
println(pair._2)
```

Výpis 3: Příklad n-tice

K elementům N-tic se nepřistupuje stejně jako v seznamech. A to z toho důvodu, že metoda seznamu apply vždy vrátí ten stejný typ, ale každý element N-tice může být typu jiného. _1 může mít jeden návratový typ, _2 zase jiný atd. _N hodnoty jsou jednozákladové namísto nula- základových, protože začínat s hodnotou 1 je tradice, která pochází z jiných jazyků se staticky typovanými n- ticemi, jako je Haskell a ML.

2.8 Objekty typu Singleton

Třídy ve Scale nemohou mít statické členy. To je jeden z důvodů, proč je Scala objektově-orientovanější než jazyk Java. Namísto statických členů Scala používá objekty singleton. Definice objektu typu singleton vypadá úplně stejně jako definice třídy, jen se místo klíčového slova `class` použije klíčové slovo `object`. Pokud je jméno singleton objektu stejné jako jméno třídy, nazýváme jej přidruženým objektem třídy. Takováto třída i objekt musí být definovány ve stejném zdrojovém souboru. Třídu nazýváme přidruženou třídou objektu typu singleton. Třída i její přidružený objekt mohou vzájemně sdílet privátní složky.

Objekt typu singleton, který nesdílí stejné jméno se jménem své přidružené třídy, se nazývá samostatný objekt. Samostatné objekty můžeme využít k mnoha účelům, včetně shlukování vzájemně propojených utilizačních metod dohromady, nebo definovat vstupní bod Scala aplikace.

Jeden rozdíl mezi třídami a singleton objekty je ten, že singleton objekty nemohou přebírat parametry, kdežto třídy ano. Jelikož nemůžeme vytvořit instanci singleton objektu pomocí klíčového slova `new`, není možné mu parametry předat. Každý singelton objekt je implementován jako instance *syntetické třídy* odvozené ze statické proměnné[3].

[3]Jméno syntetické třídy se skládá ze jména objektu a znaménka dolaru. Syntetická třída pro singleton objekt *ChecksumAccumulator* je tedy *ChecksumAccumulator$*.

2.9 Aktéři a souběžnost

Mnohdy je při tvorbě programu zapotřebí provádět části kódu nezávisle na sobě. Jde o tzv. paralelismus. Java obsahuje pro tyto účely ve své specifikaci vlákna. Scala rozšiřuje standardní podporu paralelismu z Javy o aktéry. Aktéři poskytují model souběžnosti, se kterým je jednodušší pracovat a můžeme se díky němu vyhnout mnoha nepříjemnostem při použití standardního modelu souběžnosti z Javy. Aktér je entita podobná vláknům, která má mailovou schránku pro obdržené zprávy. Abychom mohli implementovat, musíme dědit ze třídy *scala.actors.Actor* a implementovat metodu *act*.

```
import scala.actors._

object SillyActor extends Actor {
    def act() {
        for (i <1to 5) {
            println ("I'm acting!")
            Thread.sleep(1000)
        }
    }
}
```

Výpis 4: Akteří a jejich implementace

Tento aktér vytiskne pětkrát za sebou zprávu jako řetězec a skončí.

2.10 Scala aplikace

Abychom mohli úspěšně spustit program ve Scale, musíme uvést jméno samostatného objektu typu singleton (viz podkapitola 2.8) s metodou *main*, která přebírá jeden parametr *Array[String]* a má výsledný typ *Unit*. Jakýkoliv samostatný objekt s metodou main se správnou signaturou může sloužit jako vstupní bod aplikace. Příklad můžeme vidět ve výpise 5.

```
// Soubor Summer.scala
import ChecksumAccumulator.calculate

object Summer {
    def main(args: Array[String]) {
        for (arg <args)
            println (arg +": "+ calculate(arg))
        }
    }
```

Výpis 5: Aplikace s metodou main

3 Webový framework Lift

Lift je v současné době nejrozšířenějším frameworkem pro programovací jazyk Scala [3]. Framework Lift nabízí velké množství funkcí, které z důvodu rozsahu této práce nemohu uvést všechny, ale vybral jsem pouze některé z nich.

3.1 Úvod do Liftu

K vytváření aplikací ve frameworku Lift používám v této práci:

- Java 1.6 JDK
- Apache Maven 3.0.3
- GlassFish Server 3.0 (Aplikační open source server)
- Programovací editor: NetBeans IDE verze 6.9.1

Spojování (binding) je základní koncept systému šablon v Liftu. Téměř na vše na úrovni HTML a XML lze pohlížet jako na sérii zanořených spojení.

Třída *Boot* je zodpovědná za nastavení a strukturu frameworku Lift. Je vždy umístěna v balíčku *bootstrap.liftweb* a je vyobrazena níže:

```
package bootstrap.liftweb
import net.liftweb._ util._
import net.liftweb.http._
import net.liftweb.sitemap._
import net.liftweb.sitemap.Loc._
import Helpers._

class Boot {
  def boot {
    // where to search snippet
    LiftRules.addToPackages("demo.helloworld")
    // Build SiteMap
    val entries = Menu(Loc("Home", List("index"),
    "Home")) :: Nil
    LiftRules.setSiteMap(SiteMap(entries:_*))
  }
}
```

Výpis 6: Struktura třídy Boot

V metodě *boot* jsou dva základní konfigurační elementy. První z nich je metoda *LiftRules.addToPackages*. Říká Liftu, aby své vyhledávání zaměřil do balíčku *demo.helloworld*. To znamená, že bychom útržky hledali v balíčku *demo.helloworld.snippets*, pohledy zase v balíčku *demo.helloworld.views*, a tak dále. Pokud máme více než jednu hierarchii (několik balíčků), zavoláme *addToPackages* několikrát za sebou. Druhou položkou ve třídě Boot je nastavení *SiteMenu*.

3.2 Základní vlastnosti Liftu

Při zpracovávání požadavků v Liftu je na prvním místě zachycení a obsluha HTTP požadavku [8]. Dříve Lift využíval ke zpracování příchozích požadavků *Servlet* [9]. Dnes je místo něj využívána instance *Filter*, která umožňuje kontejneru zpracovávat všechny požadavky, které Lift zpracovávat neumožňuje (konkrétně ty se statickým obsahem). Filter se chová jako tenký wrapper na vrchní straně existujícího *LiftServletu* (který stále vykonává veškerou práci). Soubor web.xml by tedy měl specifikovat filter a ne servlet (ukázáno na výpise 7).

Konkrétně mapování filtru (řádky 13- 16) nám říká, že *Filter* je zodpovědný za všechno. Když filtr obdrží požadavek, zkontroluje sadu pravidel aby zjistil, zda ho může zpracovat. Pokud je požadavek jedním z těch, které je Lift schopen zpracovat, tak filtr předá požadavek ke zpracování interní instanci *LiftServlet*. V opačném případě je požadavek zřetězen a je povoleno jeho zpracování kontejnerem.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
... DTD here ...
<web-app>
    < filter >
        < filter -name>LiftFilter</filter-name>
        <display-name>Lift Filter</display-name>
        <description>The Filter that intercepts lift
        calls</description>
        < filter -class>net.liftweb.http. LiftFilter
        </ filter -class>
    </ filter >
    < filter -mapping>
        < filter -name>LiftFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </ filter -mapping>
</web-app>
```

Výpis 7: Nastavení LiftFilteru v souboru web.xml

3.2.1 Renderování se šablonami

Šablony tvoří stěžejní část flexibility a síly Liftu. Šablona je XML soubor, který obsahuje tagy specifické pro Lift a samozřejmě také veškerý obsah, který chceme v aplikacích vrátit uživateli. Lift obsahuje již zabudované tagy pro specifické akce, které mají formu `<lift:snippet_name/>` . Lift taktéž umožňuje tvorbu vlastních tagů, kterým se říká *útržky* (z angl. *snippets*). Tyto uživatelsky definované tagy jsou přilinkovány přímo ke Scala metodám, které umí zpracovávat obsahy tagů útržků, nebo vygenerovat jejich vlastní obsah od základů. Ve výpise 8 je ukázána jednoduchá šablona.

```
< lift :surround with="default" at="content">
<head><title>Hello!</title></head>
< lift :Hello.world />
```

```
</ lift :surround>
```

Výpis 8: Ukázka šablony Liftu

3.3 Tvorba formulářů

Lift podporuje kromě standartního zpracovávání formulářů pomocí GET/POST i pomocí technologií AJAX a JSON [11]. Nicméně v této kapitole bych se rád soustředil na standartní HTML formuláře. Formulář v Liftu je vyprodukován pomocí útržku, který obsahuje atribut *form*. Tento atribut přebírá hodnoty GET a POST a umožňuje kódu útržku vložit tagy formuláře z vnějšku útržku HTML.

```
< lift :AddEntry.addentry form="POST"
multipart="true">
<div class="column span-24">
   <h3>Entry Form</h3>
<div id="entryform">
```

Výpis 9: Struktura formuláře

3.4 SiteMap

SiteMap je velmi silnou součástí Liftu, který vykonává prakticky přesně to, co napovídá název: poskytne našim stránkám mapu (menu). Kromě této základní funkcionality ovšem nabízí řadu dalších možností využití:

- Mechanismy kontroly přístupu, které kontrolují, zda je funkční stránka, na kterou je odkazováno
- Seskupení položek menu, tudíž můžeme jednoduše zobrazit části menu kdekoliv chceme
- Umožňuje vytvářet zanořená menu, tedy můžeme zde mít hierarchii
- Přepisování požadavků

3.5 Vrstva objektově- relačního mapování (ORM)- Mapper a Record

Při práci s webovými aplikacemi se bezesporu setkáme s problémem, kam uložit data. Když zpracováváme data od uživatelů, začneme se potýkat s problémy jako je kódování formulářů, validace a výdrž zpracovávat data. Zde přicházejí na řadu frameworky *Mapper* a *Record*, které nám umožňují provádět veškerou manipulaci s daty, kterou potřebujeme. Mapper je původní perzistentní framework jazyka Lift a způsobem ukládání dat je úzce svázán s JDBC. Record je novou refaktorizací Mapperu, tedy nezáleží, zda chceme naše data ukládat přes JDBC, JPA, nebo dokonce XML.

Kromě *Mapperu* a *MetaMapperu* obsahuje framework i třetí rys, kterým je *MappedField*. Ten zajišťuje třídě funkcionalitu jednotlivých polí. Definujeme zde jak jednotlivé validátory, tak rovněž transformující se filtry a vyplněná jména.

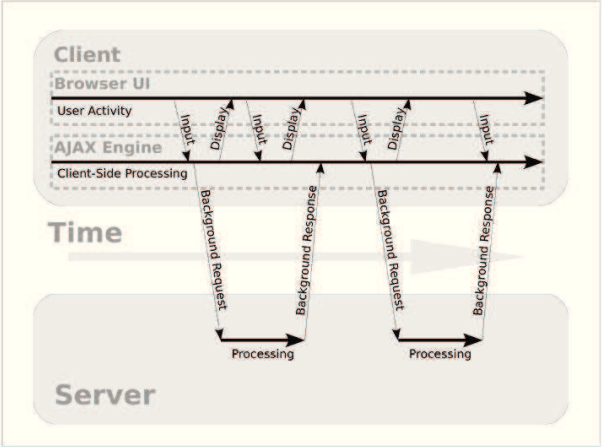
Poznámka 3.1 Jelikož Mapper není v Mavenu přímo obsažen a jedná se o oddělený modul, přidal jsem závislost do souboru pom.xml, abych k němu mohl přistupovat.

```
<dependency>
  <groupId>net.liftweb</groupId>
  <artifactId>lift-mapper</artifactId>
  <!-- or 1.1-SNAPSHOT, etc, to match your project -->
  <version>1.0</version>
</dependency>
```

Výpis 10: Přidání závislosti pro modul Mapper

3.6 Podpora AJAXu a Comet

Pomocí jazyka AJAX [12] můžeme vytvářet dynamické webové stránky a zpříjemnit tak uživatelům návštěvu našich stránek. Podpora tohoto jazyka je v Liftu na pokročilé úrovni. Na obrázku 1 je ukázán způsob práce Ajaxu (obrázek převzat z [3]).



Obrázek 1: Aplikační model AJAXu.

Hlavním rozdílem je to, že klasický *SHtml* generátor zpětnovazebně vrací *Any*, AJAX musí vrátit *JsCmd*. Důvodem je to, že návrat ze zpětného volání je samo o sobě volání na straně klienta, které může být použito na aktualizaci klientského obsahu.

Druhým důležitým aspektem podpory AJAXu je to, že Lift poskytuje robustní systém podřízenosti AJAXu. Na příklad Lift poskytne svůj vlastní JavaScript, který se stará o znovuobnovení procesu, pokud vyprší čas předání. Můžeme ovládat délku trvání vypršení

času a poté zkusit spustit proces znovu pomocí *LiftRules's ajaxPostTimeout* (v milisekundách) nebo proměnných *ajaxRetryCount*.

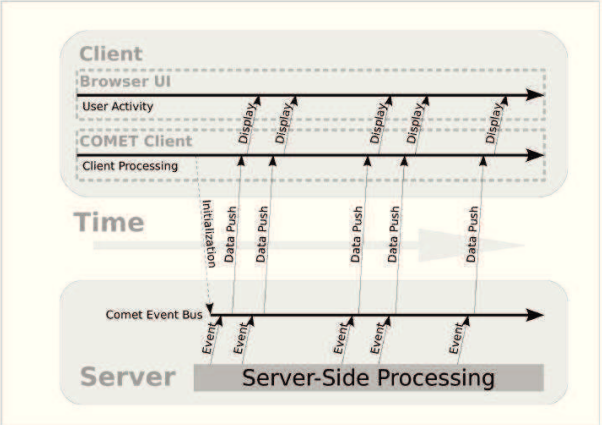
Třetím důležitým hlediskem podpory jazyka AJAX je velmi jednoduché povolení jazyka. Lift se automaticky stará o přidání potřebných JavaScriptových knihoven k našim šablonám v době vytváření a rovněž nastaví zpětné odeslání. Defaultně je odeslání nastaveno relativně k cestě `/ajax_request` v kontextu webu, nicméně Lift nám umožňuje přenastavení pomocí *LiftRules.ajaxPath*.

Posledním důležitým aspektem je flexibilita, kterou knihovna poskytuje. Kromě standardních formulářových prvků, které mohou být typu *AJAXfied*, Lift rovněž poskytuje metodu *SHtml.ajaxCall*, která vytváří *JsExp* jež můžeme použít přímo na jakýkoliv element. Navíc nám to umožňuje vytvořit pro naši odesílací funkci v JavaScriptu argument typu *String*, takže máme plný přístup k datům na klientské straně.

```
import _root_.net.liftweb.http.SHtml._
import _root_.net.liftweb.http.js.JE._
import _root_.net.liftweb.http.js.JsCmds._
def myFunc(html: NodeSeq) : NodeSeq = {
    bind("hello", html, "button" ->
    ajaxButton(Text("Press me"),
{() =>
     println ("Got an Ajax call")
    SetHtml("my-div", Text("That's it")) })
}
```

Výpis 11: Použití jazyka AJAX v Liftu

Pokud se máme bavit také o Comet, tak si musíme říct, že nejde o technologii, ale o techniku, která umožňuje webové aplikaci protlačit zprávy ze serveru ke klientovi. Využívá tzv. dlouhého poolování u HTTP požadavku v pozadí, který umožňuje serveru směrovat data do prohlížeče bez nutnosti vyžádání dalších požadavků. Na obrázku 2 je ukázán způsob práce Comet (obrázek převzat z [3]).



Obrázek 2: Aplikační model Comet.

3.7 Webové služby

V dnešní době mnoho webových aplikací nabízí API, které umožňuje ostatním lidem rozšířit funkcionalitu služby. API je množina funkcí, které jsou k dispozici třetím stranám a umožňují znovupoužití prvků aplikace.

K tomu abychom mohli sestavit webovou službu, musíme znát práci s požadavky a odpověďmi protokolu HTTP.

```
]> curl -v http://demo.liftweb.net/
* About to connect() to demo.liftweb.net port 80 (\#0)
* Trying 64.27.11.183... connected
* Connected to demo.liftweb.net (64.27.11.183) port 80 (\#0)
> GET / HTTP/1.1
> User-Agent: curl/7.19.0 (i386-apple-darwin9.5.0)
 libcurl /7.19.0 zlib /1.2.3
> Host: demo.liftweb.net
> Accept: */*
```

Výpis 12: Klientský HTTP požadavek

3.7.1 Definování REST aplikace

Roy Fielding definoval REST ve své dizertační práci [16] a definoval hlavní princip architektury jako jednotné rozhranní pro jednotlivé zdroje. „Zdroje“ odkazují na části informací, které jsou pojmenovány a jsou jsou určitým způsobem reprezentovány. Jednotné rozhranní je proloženo množinou následujících omezení (vlastnosti popsané v této kapitole vychází z [3]):

- *Bezestavovost komunikace*: Postavena na vrchní straně protokolu HTTP, který je také bezestavový.
- *Interakce ve formě Client- Server*: Tak jako se web skládá z jednotlivých prohlížečů komunikujících se servery, tak i RESTové služby umožňují strojům nebo aplikacím komunikovat se servery stejným způsobem.
- *Podpora vyrovnávací paměti*: REST používá hlavičky vyrovnávací paměti HTTP protokolu ke cachování zdrojů.

Tyto služby jsou společné jak pro webové, tak pro *RESTful* služby. REST služby přidávají dodatečná omezení, která se týkají interakce se zdroji:

- *Pojmenování*: Zdroj musí být identifikován a to za použití identifikátorů URL.
- *Popisné akce*: Použití HTTP akcí, GET, PUT a DELETE přesně ukazuje, která akce byla provedena na zdroji.
- *Adresovatelnost URL*: Identifikátory URL by měly být povoleny pro adresování reprezentace zdroje.

RESTful architektury obsáhnou HTTP protokol a využívají jej.

```
package com.pocketchangeapp.api

object RestAPI extends XMLApiHelper {
  def dispatch: LiftRules.DispatchPF = {
    case Req(List("api", "expense", eid), "", GetRequest) =>
      () => showExpense(eid)
    case r @ Req(List("api", "expense"), "", PutRequest) =>
      () => addExpense(r)
      // Invalid API request – route to our error handler
    case Req(List("api", x), "", _) => failure _
  }
}
```

Výpis 13: Pravidla odesílání

Nyní se podívejme na názorný příklad. Server nyní bude obsluhovat požadavky GET s metodou *showExpense* a bude zpracovávat PUT požadavky s metodou *addExpense*. Musíme si uvědomit, že používáme spojování vzorů na objekt *Req*. V požadavku PUT extrahujeme *Req* a předáváme jej jako parametr metodě *addExpense*, protože předáváme tělo XML s informací pro *Expense*.

3.8 Integrace JPA

Java Persistence API (JPA) bylo vivinuto ve velkém množství frameworků v Javě, kdy poskytuje jednoduchou vrstu pro přístup k databázi pro Java a Scala objekty. JPA bylo vytvořeno jako část specifikace Enterprise Java Beans 3 (EJB3), tak aby zjednodušilo perzistentní model. Lift sice přichází s dobře vytvořenou abstraktní vrstvou (jak jsem již zmínil v kapitole 3.5, o Mapperu a Recordu), nicméně jsou zde některé výhody, proč stojí za to se zajímat i o použití JPA.

JPa je jednoduše přístupné jak z Javy, tak ze Scaly. Pokud bychom používali Lift jako doplnění k projektu, který obsahuje rovněž Javu, JPA nám umožní použít databázovou vrstvu mezi oběma jazyky a vyhneme se tudíž duplikaci kódu. Rovněž to znamená, že pokud máme již existující projekt založen na JPA, můžeme jej jednoduše integrovat do Liftu. JPA také poskytuje větší flexibilitu při použití rozsáhlých schémat. I když Mapper poskytuje větší část funkcionality kterou potřebujeme, JPA dodatečné metody životního cyklu a metody práce s mapováním nám ulehčují zpracování při komplexnějších potřebách. JPA má lepší podporu pro spojování a vztahy mezi entitami. A rovněž nám JPA poskytuje zlepšení výkonu při caschování objektů, kdy je možno zachytit objekty, ke kterým se často přistupuje, do paměti tak, abychom se vyhnuli přílišnému zatížení databáze.

3.9 Integrace aplikací třetích stran

Často také využíváme možnost integrovat do našich aplikací knihovny třetích stran, které obsahují správu a některé možnosti, které mohou být pro uživatele atraktivní a mohou ulehčit vývoj našich projektů. Mezi příklady integrací aplikací třetích stran patří

např. populární platební systém PayPal, sociální síť Facebook, XMPP protokol (sloužící ke komunikaci v reálném čase a rovněž k zasílání správ mezi uživateli- vytvořeno Jabber open-source komunitou). Dále OpenID, které slouží k vytváření a spravování uživatelských účtů za použití zabezpečených protokolů.

3.10 Widgety

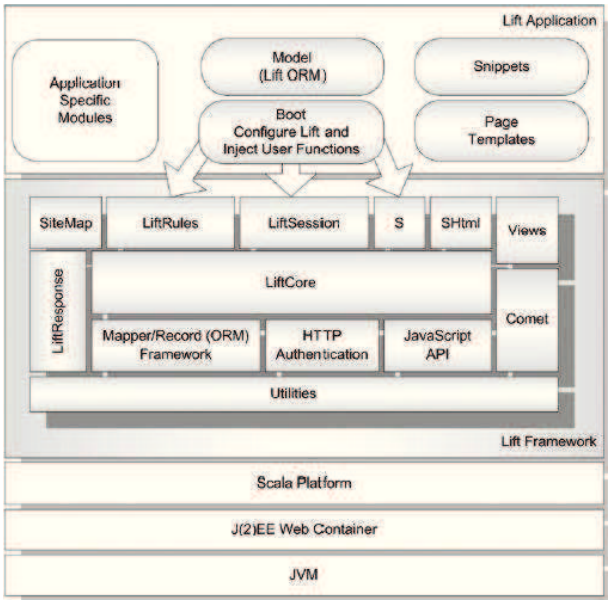
Widget je v podstatě knihovna kódů Scala a Javascriptu, které společně poskytují zabalené XHTML fragmenty, zobrazující se v klientském prohlížeči. V jiných webových frameworcích (JavaServer Faces, Struts aj.) se jim říká komponenty. Widgety vylepšují dynamické chování na straně klienta, což je tudíž dělá velmi zajímavými a používanými. Lift se snaží co nejvíce ulehčit použití widgetů, takže narozdíl od některých jiných frameworků stačí zapsat do aplikace několik řádků kódu a widget je okamžitě funkční a připraven k použití. Widgety jsou umístěny v modulu `lift-widgets`, který musíme naimportovat do souboru `pom.xml`, pokud je chceme využívat.

- TableSorter- založen na pluginu `TableSorter` z jQuery, umožňuje nám vzít existující HTML tabulku a přidat do ní možnost seřazení sloupců
- Calendar- zobrazuje kalendář ve třech formátech (měsíční, týdenní, denní), který má podobný vzhled jako kalendář z Microsoft Outlook nebo Goole kalendář; poskytují základní funkcionalitu pro zobrazení, ale je možno upravit soubory CSS a JavaScriptu, aby kalendář odpovídal na míru našim požadavkům
- RSS Feed- jak název napovídá, tento widget nám zobrazuje RSS kanály
- TreeView- přeměňuje neseřazený seznam (`<ul>`) do stromové struktury, za použití `TreeView` pluginu pro JQuery

Existují ještě některé další widgety, jako např. Gravatar a Sparklines, ale o těch se zde již zmiňovat nebudu. V Liftu máme rovněž možnost vytvářet si vlastní widgety, kde se fantazii meze nekladou. Jde o zajímavé prvky, které mohou obohatit aplikaci jak z hlediska lepšího vzhledu, tak propracovanější funkcionality.

3.11 Architektura Liftu

Na obrázku 3 jsou znázorněny hlavní komponenty a jejich umístění v celém systému (schéma architektury převzato z [3]). Jak jsem již zmiňoval v kapitole 3.2, Lift je navržen tak, aby se choval jako *Filter* sloužící jako vstupní bod aplikace. Způsob jakým využijeme zbývající části frameworku se liší u každé aplikace, podle toho jak moc je komplexní.



Obrázek 3: Architektura frameworku Lift.

4 Play framework

K danému frameworku Lift jsem si měl zvolit druhý framework a vzájemně oba frameworky porovnat. V současné době existuje řada frameworků, které jsou založené na Scale:

- Sweet- oproti Liftu mnohem jednodušší framework, založen na jazyku šablon (freemarker) [17]
- Slinky- nově vznikající framework, založen na balíčku *Scalaz* (záměrem tohoto balíčku je zahrnout funkce, které nejsou momentálně dostupné v základním Scala API) [18]
- Scalatra- následuje principy frameworku *Sinatra* (založeném na Ruby), původně znám pod názvem *Step*; založen na tzv. cestách, kterými se rozumí HTTP metoda spojena s odpovídajícím URL vzorem [19]
- Pinky- framework spojující Scalu, REST a MVC architekturu, postaven nad *Guice* (odlehčený framework pro Javu, vytvořený společností Google) a *Guice Servletem*; poskytuje podporu formulářů, domén objektů aj. [20]
- Scala Pages (SCP)- odlehčený framework pro tvorbu Scala aplikací založen na textově orientovaném enginu se šablonami (XML šablony, JSON aj.); ulehčuje vytváření aplikací jako MVC architektur; vytvořen tak, aby vyhovovala požadavkům moderních procesorů [21]

Nicméně já jsem zvolil pro porovnávání framework Play, který se mi zdál na základě zjištěných vlastností a obsáhlosti dokumentace pro praktické využití nejvhodnější.

Framework Play! [14] je původně založen na Javě, ale dnes již existuje i podpora Scaly.

4.1 Vývoj řízený pomocí testů

Vestavěný spouštěč testů nám velmi jednoduše umožňuje provádět vývoj řízený za pomoci testů. Můžeme psát všechny druhy testů, od jednoduchých Unit testů až po úplné vstupní testy a jednoduše je spouštět v prohlížeči.

4.2 Struktura projektů

- *app/* obsahuje jádro aplikace, v této složce jsou umístěny *.scala* soubory.
- *conf/* obsahuje všechny soubory k nastavení aplikace, speciálně hlavní soubor *application.conf*, soubory definic *routes*, soubor *dependencies.yml* a soubory *messages*, sloužící ke zpracování mezinárodních formátů
- *lib/* v této složce jsou umístěny všechny nepovinné knihovny Javy nebo Scaly, zabaleny jsou standardně ve formátu *.jar*

- *public/* obsahuje všechny veřejně přístupné zdroje, včetně souborů JavaScriptu, stylů a složek s obrázky
- *test/* zde jsou uloženy všechny testy aplikace

Soubor nastavení závislostí *conf/dependencies.yml* je automaticky nastaven na závislost Scaly. Můžeme se divit, kde se poděly všechny *.class* soubory. Odpověď zní nikde. Play nevyužívá žádných class souborů, namísto toho čte zdrojové kódy Scaly rovnou.

To v procesu vývoje umožňuje provádět dvě velmi důležité věci. První z nich je ta, že Play bude zachytávat jakékoliv změny, které provedeme v jakémkoliv zdrojovém Scala souboru a automaticky je při běhu znovu načte. Druhou je způsob obsluhy vyjímek. Pokud nastane v aplikaci výjimka, Play pro nás vytvoří lepší chybové zprávy a zobrazí nám přesné místo, kde k výjimce došlo.

Aby Play urychlil restart rozsáhlých aplikací, uchovává ve složce *tmp/* byte kód vyrovnávací paměti. Pokud chceme, můžeme to zrušit použitím příkazu *play clean*.

Play aplikace má na rozdíl od Scaly samotné několik vstupních bodů aplikace, pro každé URL jeden. Těmto metodám říkáme *metody akcí*. Tyto metody jsou definovány ve speciálních objektech, kterým se říká Ovladače.

4.3 Scala ovladače

Ovladače frameworku Play jsou nejdůležitější součástí jakékoliv aplikace napsané v Play. Aplikace Play Scala sdílí stejné koncepty jako klasická Play aplikace, ale užívá více funkcionální způsob popisu jednotlivých akcí.

Controller je singleton objekt jazyka Scala, podporován v balíčku *controllers* a ve třídě *play.mvc.Controller*. V jednom souboru můžeme definovat neomezený počet ovladačů. Jelikož nám Scala poskytuje podporu singleton objektů, nemusíme se již déle potýkat se statickými metodami Javy a snahou udržení schopnosti staticky odkazovat jakoukoliv akci jako třeba *show(id)* (viz ukázka kódu 14).

```
package controllers {

    import play._
    import play.mvc._

    object Users extends Controller {

        def show(id:Long) = Template("user" -> User.findById(id))

        def edit(id:Long, email:String) = {
            User.changeEmail(id, email)
            Action(show(id))
        }
    }
}
```

Výpis 14: Pravidla odesílání

Ovladač Play obvykle používá ke spuštění vygenerování odpovědi imperativní uspořádání jako *render(...)* nebo *forbidden()*. Naproti tomu na metody akcí napsané ve Scale se pohlíží jako na funkce a musí tedy vracet hodnotu. Tato hodnota je poté použita frameworkem k vygenerování HTTP odpovědi na dotaz.

Metoda akce může samozřejmě vracet několik typů hodnot, v závislosti na požadavku. Zde jsou příklady některých z nich: *Template*, *Forbidden*, *Html*, *JSon*, *Redirect*, *BadRequest* aj.

4.4 Práce s databází

Možnosti práce s databází webového frameworku Play.

4.4.1 Výběr a nastavení databáze

Pro potřeby vývoje přichází Play se samostatným SQL systémem řízení báze dat, který se nazývá *H2*. Nastavení databáze se provádí v konfiguračním souboru *application.conf* odkomentováním příslušných řádků pro konkrétní databázi (v mém případě odkomentuji pouze jeden řádek s `db=mem`). V mé aplikaci automaticky dojde k připojení k databázi, jak ukáže výpis, který se objeví na konzoli: `INFO : Connected to jdbc:h2:mem:play`.

4.4.2 Databázové připojení

Anorm je zjednodušení JDBC s minimalistickým rozhraním, které znovupoužívá dříve vytvořená Scala rozhraní (kolekce, spojování vzorů, kombinátory parserů). Jedná se o vrstvu přístupu k datům. Používá jazyk SQL aby mohl provádět požadavky na databázi a posyktuje několik API k parsování a přeměně výsledné množiny dat.

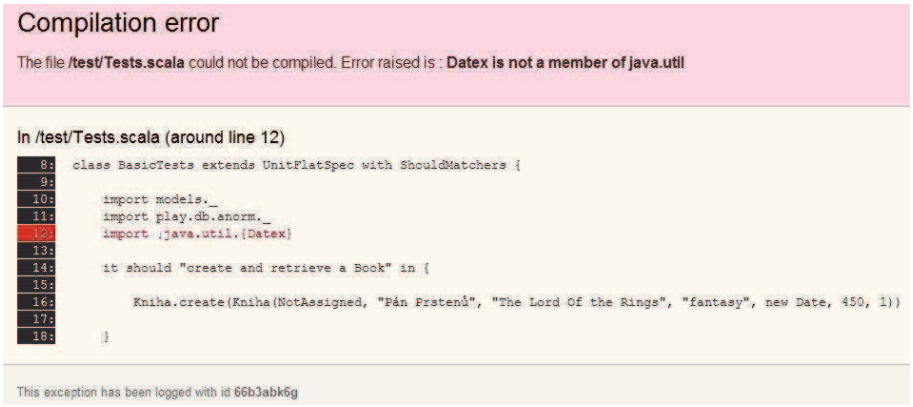
```
val postsWithAuthor:List[(Post~User)] =
   SQL(
         select * from Post p join User u on p.author_id = u.id order by p.postedAt desc
    ).as( Post ~< User * )
```

Výpis 15: Ukázka práce s databází

5 Srovnání frameworků

Play je bezestavový framework, který dodatečně poskytuje podporu Scaly. U Liftu je možné pro práci s databází používat rozličné technologie, buď Mapper a Record, nebo třeba JPA. Framework Lift používá nástroj pro správu buildů a to buďto Maven, SBT, či jiné další. Framework Play žádného nástroje pro automatizaci a správu buildů nevyužívá.

U frameworku Play jsou přehledně zobrazeny chyby, jsou ukázány v prohlížeči společně s kódem, při najetí na okno aplikace v prohlížeči (viz obr. 4).



Obrázek 4: Způsob zobrazení chybových zpráv.

Play je sice poměrně nový framework, ale komunita kolem něj se stále rychle zvyšuje. Lift je stavový framework vytvořený primárně pro Scalu. Je zde tedy velké začlenění jazyka Scala, takže není potřeba vytvářet get/set metody nebo se obávat vzájemné kompatibility mezi kolekcemi jazyka Java a Scalou. Plně využívá konceptů funkcionálně- orientovaného programování. Má výstižně psaný kód, přesto je typově bezpečný.

Lift má rozsáhlou podporu webových REST aplikací a jazyka AJAX.

5.1 Společné rysy

V případech obou frameworků se jedná o moderní, stále se rozvíjející technologie, které jsou založeny na moderním programovacím jazyce, tedy Scale. V případě Liftu i Play! je k dispozici mnoho vestavěných funkcí, které velmi usnadňují programování interaktivních webových stránek a informačních systémů a nabízejí tak velmi zajímavou variantu tvorby webových aplikací, narozdíl od již zažitých technologií. Při programování je u obou frameworků oddělena prezentační vrstva od vrstvy logické (u Liftu pomocí tzv. útržků), což značně spřehledňuje čitelnost zdrojových kódů.

5.2 Výhody Liftu

Lift je v současné době nejrozšířenějším frameworkem založeným na Scale.

Programátorská komunita je nejvíce zaměřena právě na něj a proto vývoj tohoto frameworku je rychlý. Máme více možností, jak přistupovat k databázi (buď pomocí Recordu, nebo pomocí Mapperu) a pracovat s ní. Lift nabízí velkou škálu funkcí a knihoven, které můžeme použít a pokud si vše nastudujeme, tak můžeme naprogramovat a přidat do aplikace téměř cokoliv, na co si vzpomeneme. Pro podporu asynchronního přístupu je zde možnost využití Ajaxu a Comet, dále můžeme využít JavaScriptu, integraci aplikací třetích stran (Facebook, platební systém PayPal, OpenID, otevřený internetový protokol pro psaní zpráv AMQP a mnohé další). U webových stránek je třeba se zmínit o pohodlném vytvoření položek menu pomocí funkce SiteMap. Dále je propracováno využití webových služeb v aplikaci. Jako poslední bych zmínil Lift widgety, které opět obohacují širokou škálu funkci, použitelných ve frameworku Lift. Máme zde k dispozici widget TableSorter, který nám umožňuje vytvářet html tabulky, ve kterých lze třídit údaje, například podle data vytvoření nebo jiných kritérií. Dalším widgetem je Kalendář. Zde je možno vytvořit měsíční přehled, týdenní přehled a denní přehled. Lze naprogramovat a implementovat do aplikace možnost přidávání a zaznačování událostí přímo do kalendáře. Dalším widgetem je čtečka RSS, která pomáhá renderovat vlákna RSS. Pro větší přehlednost je možno využít widgetu TreeView, kde zobrazíme položky seznamu ve stromové struktuře a mnohdy tedy více přehledněji. Samozřejmě máme možnost vytvářet si kromě zabudovaných widgetů i widgety vlastní. V liftu tedy máme k dispozici opravdu co nás napadne a lze tedy vytvářet plnohodnotné webové stránky a aplikace s obrovským množstvím předem naprogramovaných funkcí a knihoven, nemusíme tedy vše vytvářet od začátku.

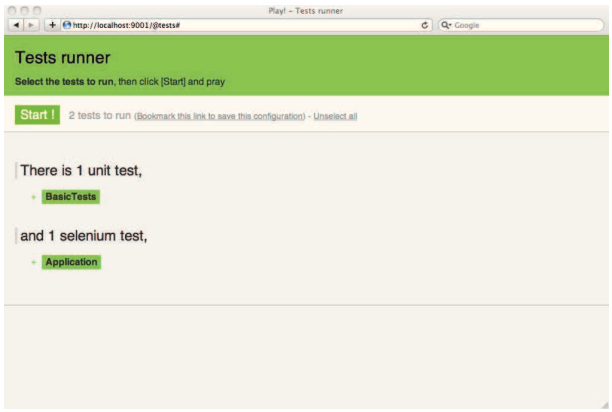
5.3 Nevýhody Liftu

Když chceme programovat v tomto frameworku, musíme poměrně dobře znát všechna úskalí Scaly a abychom mohli plně využívat všeho, co nám Lift nabízí, musíme nastudovat velké množství dokumentace. Není úplně vhodný pro začínající programátory ve Scale. Pro menší aplikace je méně vhodný, než framework Play. Překlad trvá delší dobu než u Play a jak již bylo zmíněno výše, zobrazování chyb také není tou nejsilnější stránkou frameworku.

5.4 Výhody Play

Tento framework je poměrně nový, ale jeho vývoj se neustále žene kupředu a přidávají se nové vlastnosti a funkce. Za velké výhody bych považoval jak rychlost a automatizovanost překladu při spuštění projektu (která mi je sympatičtější, než u Liftu), tak i zpracovávání jednotlivých testů, které jdou spustit snadno z prohlížeče a na ovládání a jejich spouštění si programátor rychle zvykne (viz. obrázek 5, který je převzat z [14]).

Pro vývoj jednodušších aplikací je díky své rychlosti a pružnosti framework Play lepší volbou, nemusíme se učit tolik vlastností, jako u frameworku Lift, tedy slouží rovněž pro uživatele, kteří se Scalou a programování webových stránek začínají.



Obrázek 5: Spouštění testů v prohlížeči.

5.5 Nevýhody Play

Tento framework je stále poměrně nový a mnoho funkcí v něm momentálně není naimplementováno, tudíž je třeba vždy najít nějaký jiný způsob, jak si s daným problémem při tvoření aplikací poradit. Není zde podpora Comet, dále zde nejsou naimplementovány žádné pomocné funkce a knihovny pro využití ajaxu. Vše musíme tedy programovat a implementovat od začátku. Pokud předpokládáme, že naše aplikace bude rozsáhlá, bude mít větší množství stránek a bude potřeba průběžně přidávat ještě další funkce, je na místě uvažovat spíše o použití frameworku Lift.

6 Navržená aplikace

Na tomto místě popisuji mnou vytvořenou a naprogramovanou aplikaci. Kompletní aplikace se všemi zdrojovými kódy je přiložena na CD odevzdávaném společně s bakalářskou prací- přílohy A a B. Zaměřuji se na vytvoření informačního systému knihovny, kterou budou moci využívat návštěvníci webových stránek pro vyhledávání titulů a autorů, a pracovníci knihovny pro přidávání a editaci stávajících položek. Mají možnost přidávat a editovat údaje o stávajích čtenářích a přidávat nové zájemce do systému. Dále je možnost zobrazení měsíčního kalendáře, pro lepší přehlednost. Pomocí Ajaxu je řešen ukazatel času na stránkách.

6.1 Funkční požadavky

6.1.1 Vstupy

Knihovní IS bude obsahovat údaje o jednotlivých knižních titulech. Bude zde uvedeno jedinečné identifikační číslo a rovněž název knihy (český název a název originálu), v kterém roce byl daný titul vydán a rovněž informace o počtu stran a žánru dané knihy. Dále budou zaevidováni jednotliví členové knihovny, kteří si budou chtít půjčovat jednotlivé tituly (tedy čtenáři). Každý čtenář musí při registraci vyplnit své celé jméno, adresu bydliště, poštovní směrovací číslo a e- mail, na který mu budou zasílány případné upomínky o vrácení titulu a podobně. Každému členovi bude rovněž přiřazeno jedinečné identifikační číslo.

Čtenáři budou moci prostřednictvím webových stránek tedy vyhledávat dostupné knihy z knih v systému a budou moci si je rezervovat. Přístupová práva mají omezena, nemohou editovat stávající exempláře knih ani zobrazovat uživatel. Noví zákazníci se budou moci registrovat pomocí registračního formuláře. Knihovníci budou moci zobrazit aktuální seznam všech uživatelů knihovny s informacemi o jejich výpůjčkách a případných upomínkách. V systému je vytvořeno administrační rozhranní pro knihovníky na editaci všech záznamů.

6.1.2 Funkce systému

V systému bude možno zobrazovat seznam čtenářů, seznam knih v knihovně, dále informace o knihovně obecně. Je možno si rezervovat jednotlivé exempláře, u každé rezervace musí být uveden nějaký člen, který si chce tituly rezervovat a identifikace daného exempláře. Upomínky musí obsahovat vždy číslo upomínky a rovněž číslo zaevidovaného člena, aby bylo jednoznačně dáno, kdo kterou upomínku dostal.

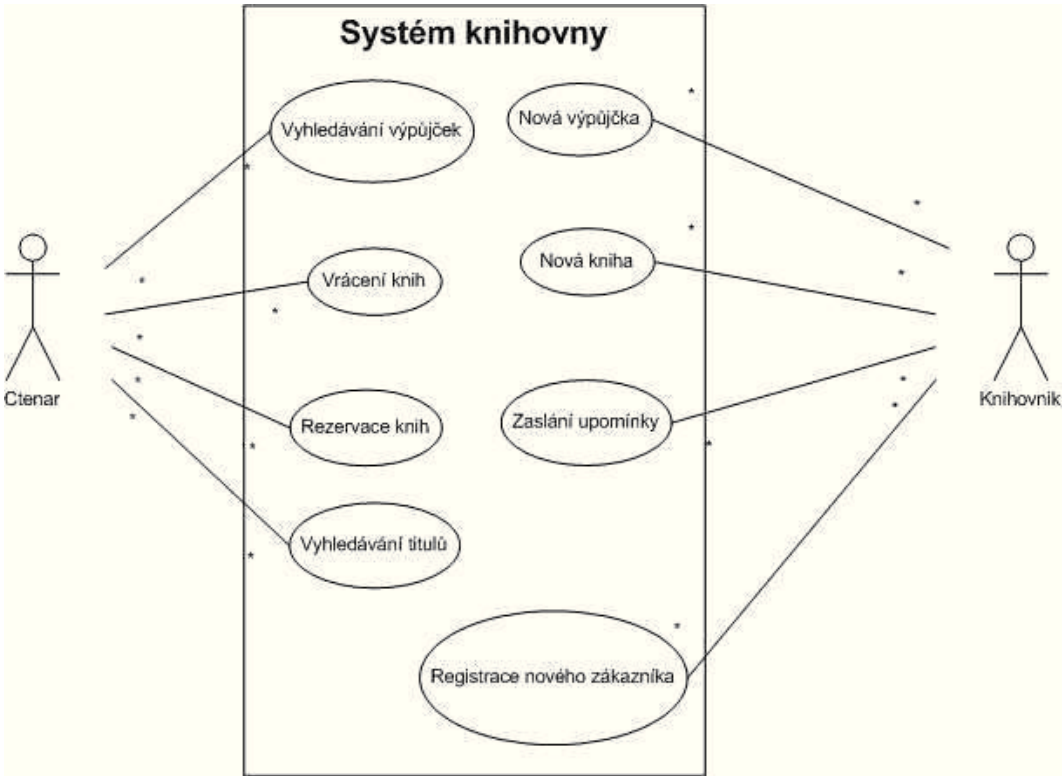
6.2 Nefunkční požadavky

U knihovního IS se předpokládá moderní a příjemné uživatelské prostředí, aby s ním mohli jednotliví zaměstnanci knihovny jednoduše pracovat. Informační systém je určen pro menší městskou knihovnu, je tedy středně velký. Čtenáři mohou navětívit webové stránky buďto z domova, nebo přímo z budovy knihovny, k dispozici by bylo několik

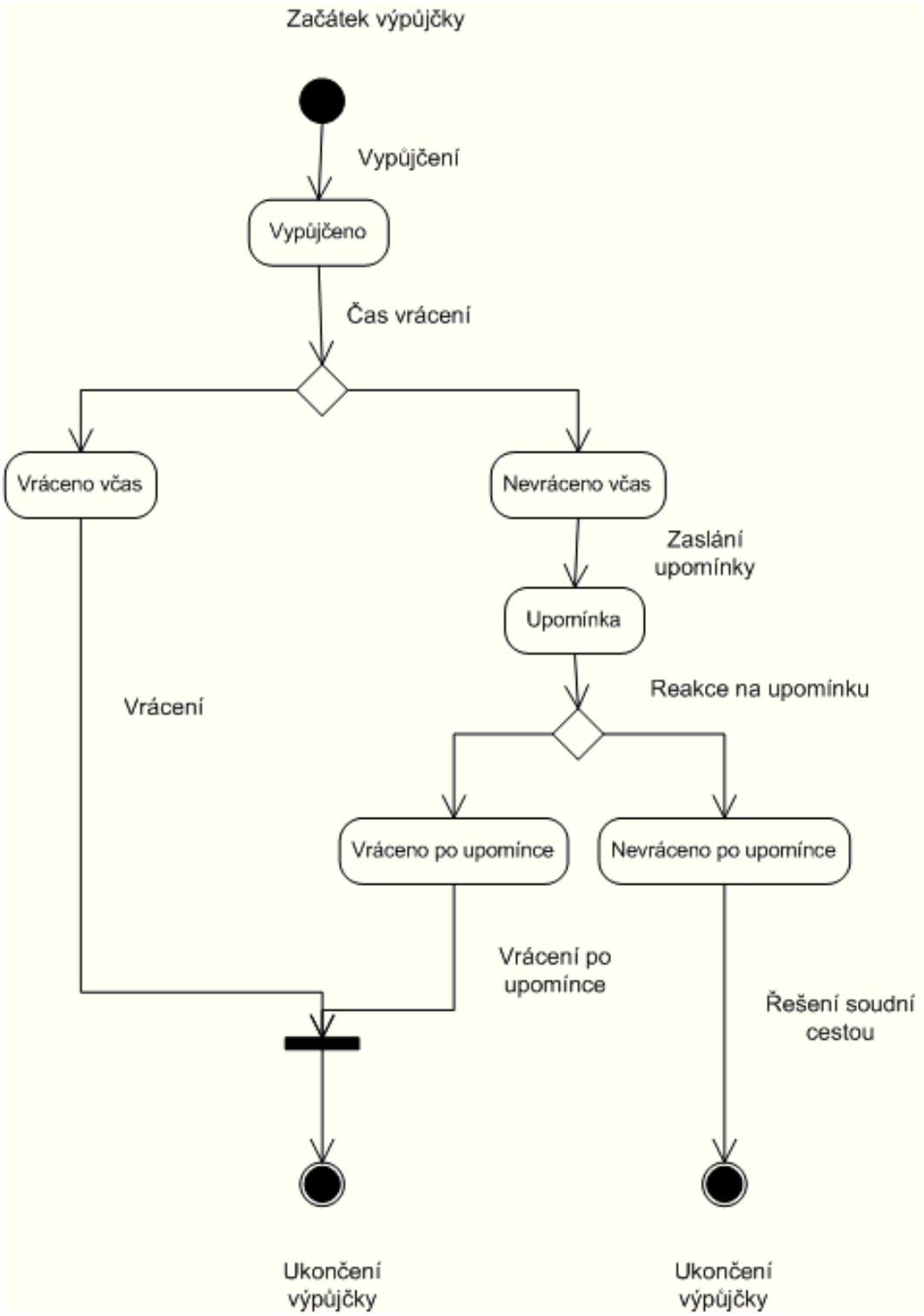
počítačů s přístupem na internet. V aplikaci jsem se soustředil spíše na ukázání některých funkcí frameworku Lift, než na vzhled webových stránek, který by pak šel samozřejmě dotvořit, například přidáním rozličných kaskádových stylů.

6.3 Konceptuální analýza

Zde zobrazuji diagram mnou navrženého systému. Schéma zobrazuji použitím diagramů jazyka UML [13].



Obrázek 6: Diagram případu užití.



Obrázek 7: Diagram aktivit u výpůjček.

6.4 Specifika řešení v jednotlivých frameworcích

Registrace a zobrazení uživatelů
Uživatelé mají možnost se při první návštěvě stránek zaregistrovat, poté už stačí jen přihlášení. Rovněž lze poslat uživateli na email heslo, pokud jej náhodou zapomene. Přidána validace jednotlivých položek při vytvoření nového čtenáře.

Lift - odkaz ke stažení knihovního řádu
Zde je použita metoda Shtml.link. Tento zpracovaný snippet je potom volán z webové stránky.

Lift - Stránkování
Pro stránkování použit template wizard, který je upraven, aby při dosažení konce stránky automaticky začal číslovat na stránce další a do zobrazení přidal tlačítko Next, kterým se přejde na další část výpisu záznamů z tabulky databáze.

Lift - Tvorba databáze
Jako databáze použita nativní databáze H2 (metoda pro spuštění databáze uložena ve vstupním bodě aplikace Boot.scala). Pro vytvážení jednotlivých objetků použit Mapper, kde jsou nastaveny jednotlivé atributy sloupců tabulky. Rovněž nastavey cizí klíče přes metodu MappedLongForeignKey. Po přepsání definice validSelectValues možno zobrazit veškeré cizí klíče, které jsou k dispozici prostřednictvím selectBoxu.

Zobrazení času v Ajaxu-Lift - v html stránce je volána vnitřní metoda TimeNow, která odkazuje na útržek (snippet), obsahující metodu pro obsluhu a zobrazení času. V metodě se vyrenderuje aktuální čas a ten je pak pomocí ajaxového volání zobrazen na stránce.

Zobrazení času v Ajaxu-Lift
- vyřešeno obdobně, akorát obsluha musela být vyřešena jinou vnitřní logikou

Tvorba menu-Lift
- vytvořeno přes nástroj SiteMap, jak už bylo zmíněno v úvodu práce. Lze nastavit možnosti zanořování menu a také nastavit, kdy se jaká část menu má objevit.

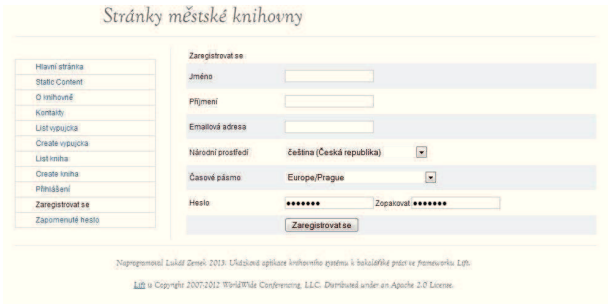
Tvorba menu- Play
- pomocí šablony XHTML

CRUD operace-Lift
Použit balíček CRUDify, který umožňuje práci s tabulkou. Editace, vyhledání, smazání a vytvoření nových záznamů.

Testování- Lift
Testování obtížnější, řešeno za pomoci editoru.

Testování- Play
Využity vestavěné Selenium testy, spouštěné v prohlížeči

Další funkce frameowrků popsány v uživatelské příručce v příloze C bakalářské práce společně s uvedenými zdrojovými kódy.



Obrázek 8: Ukázka registrace nového uživatele do systému.

Obrázek 9: Zobrazení cizího klíče v tabulce nových výpůjček.

7 Závěr

Dle zadání byly popsány jednotlivé prvky programovacího jazyka Scala a kombinace objektového a funkcionálního přístupu, které tento jazyk využívá a na jakém je postaven. Uvedl jsem informace o v současné době nejrozšířenějším webovém frameworku Lift, který je postaven na Scale a plně využívá jejích možností. Popsal jsem architekturu Liftu a rovněž jsem uvedl způsob vytváření webových aplikací za pomoci tohoto frameworku. Jako druhý webový framework jsem si vybral framework Play. Popsal jsem rovněž jeho hlavní rysy, strukturu projektů a způsob práce s ním. Také jsem popsal způsob vývoje pomocí testů, které jsou poté spouštěny v prohlížeči. Navrhl jsem a implementoval ukázkovou aplikaci, na které jsem se snažil ukázat výhody a nevýhody každého z frameworků a využít jejich potenciál k tvorbě aplikací. V poslední kapitole jsem shrnul závěry a poznatky, které jsem učinil při vytváření této ukázkové aplikace.

Play i Lift jsou frameworky které mají velké množství funkcí a mají uživatelům oba co nabídnout. Framework Lift bych spíše použil při vývoji aplikací o kterých víme, že se budou časem rozrůstat a budou zpracovávat velký provoz. Díky velké integraci se Scalou je potřeba dobře ovládat jazyk Scala. Framework Play bych použil spíše při vytváření menšího počtu jednodušších webových aplikací.

Na řešené téma lze navázat následovně. Je možno se hlouběji zaměřit na způsoby objektově-relačního mapování v jednotlivých frameworcích a implementovat webové služby. Webové frameworky zmíněné v této práci by se daly dále porovnat s nějakými dalšími frameworky, např. Scalatra nebo Sweet. Další možností je vylepšit prezentační vrstvu webových stránek a využít možností již předdefinovaných nebo nově vytvořených šablon.

Lukáš Zemek

8 Reference

[1] DocForge Wiki, *Web Application framework*, 2010.

[2] M. Odersky, L. Spoon, B. Venners, *Programming in Scala (Second edition)*, California: Artima Press, 2011.

[3] D. Chen-Becker, M. Danciu, T. Weir, *The Definitive Guide to Lift: A Scala-based Web Framework*, New York: Apress, 2009.

[4] D. Pollak, *Beginning Scala*, New York: Apress, 2009.

[5] M. Odersky, *The Scala Language Spceification Verison 2.8*, Switzerland: Draft, 2010.

[6] D. Pollak, *Simply Lift, http://simply.liftweb.net/*, 2011.

[7] D. Chen-Becker, M. Danciu, T. Weir, *Exploring Lift, Lift 2.0 Edition, http://exploring.liftweb.net/*, 2008-2011.

[8] R. Fielding, J. Gettys, J. Mogul, P. Leach, *Hypertext Transfer Protokol, http://www.w3.org/Protocols/rfc2616/rfc2616.html*, 1999.

[9] D. Coward, Y. Yoshida, *Java(TM) Servlet API Specification (Version 2.4)*, Sun Microsystems, Inc., 2003.

[10] J. Gosling, B. Joy, G. Steele, G. Bracha, *The Java(TM) Language Specification (Third Edition)*,Addison-Wesley, 2005.

[11] D. Crockford, *JSON: The Fat-Free Alternative to XML*, Presented at XML 2006 in Boston, 2006.

[12] A. T. Holdener III, *AJAX: The Definitive Guide*, O'Reilly, 2008.

[13] *OMG Unified Modeling Language Infrastructure, V2.1.2*, 2007.

[14] *Play- Scala framework documentation, http://scala.playframework.org/documentation/scala-0.9/home*, 2011.

[15] E. Truyen, W. Joosen, B. Norregaard Jorgensen, P. Verbaeten, *A Generalization and Solution to the Common Ancestor Dilemma Problem in Delegation-Based Object Systems*, Proceedings of the 2004 Dynamic Aspects Workshop (103–119), 2004.

[16] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*,Doctoral dissertation, University of California, Irvine, 2000.

[17] Sweet framework documentation, *http://code.google.com/p/sweetscala/*, 2011.

[18] Documentation for Slinky framework and Scalaz, *http://code.google.com/p/scalaz/*, 2011.

[19] Scalatra documentation, *https://github.com/scalatra/scalatra,* 2011.

[20] Pinky documentation, *https://github.com/pk11/pinky,* 2011.

[21] Scala Pages, *SCP framework documentation, http://www.thomasknierim.com/scala-pages-web-framework/,* 2011.

9 Přílohy

Zde je uveden seznam příloh na disku, který je přiložen k této bakalářské práci:

Příloha A – ukázková aplikace implementovaná ve webovém frameworku Lift
Příloha B – ukázková aplikace implementovaná ve webovém frameworku Play
Příloha C – dokumentace k implementované webové aplikaci